Exhibit 20 to Amended Complaint
Intellectual Ventures I LLC and Intellectual Ventures II LLC

[redacted]

Example Southwest Count VII Systems and Services
U.S. Patent No. 7,712,080 ("the '080 Patent")

The Accused Systems and Services include without limitation Southwest systems and services that utilize Spark; all past, current, and future systems and services that operate in the same or substantially similar manner as the specifically identified systems and services; and all past, current, and future Southwest systems and services that have the same or substantially similar features as the specifically identified systems and services ("Example Southwest Count VII Systems and Services" or "Southwest Systems and Services").[1]

On information and belief, the Southwest Systems and Services use Spark in Southwest's private cloud(s). For example, Southwest posts, or has posted, job opportunities that require familiarity with Spark technology.

See https://www.linkedin.com/in/meghanaa-malleboina/, job profile of data engineer, stating that they utilized Pyspark for distributed data processing, leveraging RDD and DataFrame APIs for large-scale data transformations and implementing window functions for complex aggregations. (Last accessed on 10/8/2024).

See https://www.linkedin.com/in/sudhir-saxena-45b9a524/, job profile of senior data engineer stating that they use PySpark to create source to target mapping to design Data pipeline. (Last accessed on 10/8/2024).

See https://www.linkedin.com/in/girishkumar81/, job profile of senior Azure data engineer who uses Apache Spark for his position at Southwest. (Last accessed on 10/8/2024).

See https://www.linkedin.com/in/amaranath-repati-a120bb22b/, job profile of data engineer stating the use of Spark. (Last accessed on 10/8/2024).

See https://www.linkedin.com/in/kirity-ganga, job profile of data engineer stating the use of Spark. (Last accessed on 10/8/2024).

---

[1] For the avoidance of doubt, Plaintiffs do not accuse the public clouds of Defendant, to the extent those services are provided by a cloud provider with a license to Plaintiffs' patents that covers Defendant's activities. [redacted] Plaintiffs do not accuse the public clouds of Defendants if those services are provided by a cloud provider with a license to Plaintiffs' patents that covers Defendants' activities. [redacted]

[redacted]

As another example, Southwest has stated that it is investing in cloud technology and has "moved about 50% of its technology" to the cloud and has indicated cloud migration is one of its areas of focus for 2024 and beyond.

Source: https://www.phocuswire.com/southwest-airlines-cio-tech-investment.

On information and belief, other information confirms Southwest uses Spark technology.

A friend of mine recently had a data engineering interview at Southwest Airlines. The interview focused on real-world scenarios, especially working with large datasets. One question he faced was particularly challenging and required a good understanding of flight data analysis.

Source: https://medium.com/towards-data-engineering/inside-a-southwest-airlines-data-engineering-interview-2f33ed964b5c.[2]

[2] Unless otherwise noted, all sources cited in this document were publicly accessible as of the filing date of the Amended Complaint.

[redacted]

U.S. Patent No. 7,712,080 (Claim 9)	
Claim(s)	**Example Southwest Count VII Systems and Services**
[9.pre]. A distributed parallel computing system, having at least one memory area and at least one processor, comprising:	To the extent this preamble is limiting, on information and belief, the Southwest Count VII Systems and Services include a distributed parallel computing system having at least one memory area and at least one processor. For example, Spark consists of a distributed parallel computing system having multiple memory and processors. **Overview** At a high level, every Spark application consists of a *driver program* that runs the user's `main` function and executes various *parallel operations* on a cluster. The main abstraction Spark provides is a *resilient distributed dataset* (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it. Users may also ask Spark to *persist* an RDD in memory, allowing it to be reused efficiently across parallel operations. Finally, RDDs automatically recover from node failures. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html. **Which Storage Level to Choose?** Spark's storage levels are meant to provide different trade-offs between memory usage and CPU efficiency. We recommend going through the following process to select one: • If your RDDs fit comfortably with the default storage level (`MEMORY_ONLY`), leave them that way. This is the most CPU-efficient option, allowing operations on the RDDs to run as fast as possible. • If not, try using `MEMORY_ONLY_SER` and selecting a fast serialization library to make the objects much more space-efficient, but still reasonably fast to access. (Java and Scala) • Don't spill to disk unless the functions that computed your datasets are expensive, or they filter a large amount of the data. Otherwise, recomputing a partition may be as fast as reading it from disk. • Use the replicated storage levels if you want fast fault recovery (e.g. if using Spark to serve requests from a web application). *All* the storage levels provide full fault tolerance by recomputing lost data, but the replicated ones let you continue running tasks on the RDD without waiting to recompute a lost partition. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.

[redacted]

U.S. Patent No. 7,712,080 (Claim 9)	
Claim(s)	**Example Southwest Count VII Systems and Services**
[9.a] at least one distributed shared variable capable of loading into the at least one memory area, wherein the at least one distributed shared variable is a single variable that includes several variables that may be physically loaded into the at least one memory area; and	On information and belief, the Southwest Count VII Systems and Services include at least one distributed shared variable capable of loading into the at least one memory area, wherein the at least one distributed shared variable is a single variable that includes several variables that may be physically loaded into the at least one memory area. For example, Spark collects large volumes of data and distributes the data across multiple nodes. Spark automatically partitions the data into Resilient Distributed Datasets (RDDs) that operate in parallel. The RDDs are partitioned across multiple nodes and processed across at least one memory area. **Resilient Distributed Datasets (RDDs)** Spark revolves around the concept of a *resilient distributed dataset* (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel. There are two ways to create RDDs: *parallelizing* an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.

[redacted]

U.S. Patent No. 7,712,080 (Claim 9)	
Claim(s)	**Example Southwest Count VII Systems and Services**
	RDDs — Partitions • RDDs are designed to contain huge amounts of data , that cannot fit onto one single machine → hence, the data has to be partitioned across multiple machines/nodes • Spark automatically partitions RDDs and distributes the partitions across different nodes • A partition in spark is an atomic chunk of data stored on a node in the cluster • Partitions are basic units of parallelism in Apache Spark • RDDs in Apache Spark are collection of partitions • Operations on RDDs automatically place tasks into partitions, maintaining the locality of persisted data Source: https://blog.k2datascience.com/batch-processing-apache-spark-a67016008167. **Parallelism** Parallelism in Spark is all about doing multiple things at the same time. The core idea is to divide the data into smaller chunks (partitions), so they can be processed simultaneously. The more partitions you have, the more tasks can run in parallel, leading to faster processing times. Source: https://dataengineerinterview.substack.com/p/a-deep-dive-into-apache-spark-partitioning.

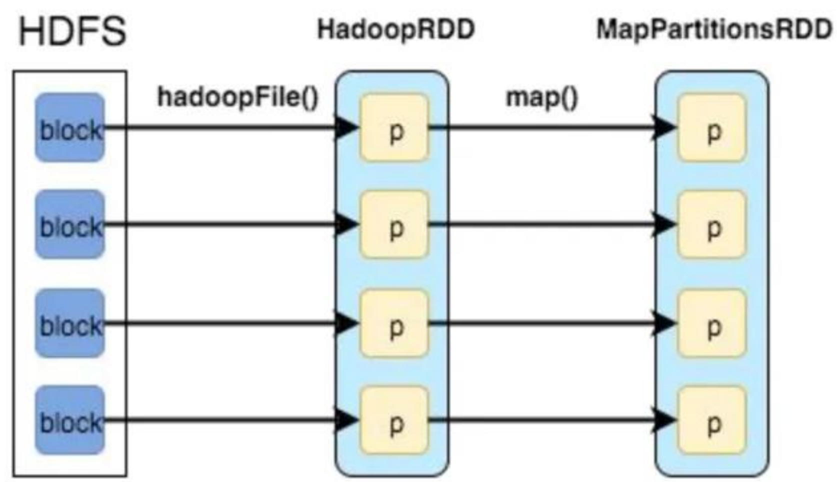
[redacted]

U.S. Patent No. 7,712,080 (Claim 9)	
Claim(s)	**Example Southwest Count VII Systems and Services**
	Imagine you have a dataset with 1 million records. If this dataset is divided into 10 partitions, Spark can launch 10 tasks to process each partition simultaneously. If the cluster has sufficient resources, these tasks will run in parallel, speeding up the process. However, if you increase the number of partitions to 100, and assuming your cluster has the resources (like CPU cores) to handle these tasks, Spark can now launch 100 tasks to run concurrently. This dramatically increases parallelism and reduces the total time taken to process the entire dataset. Source: https://dataengineerinterview.substack.com/p/a-deep-dive-into-apache-spark-partitioning. **In-Memory Processing: Spark's Secret Sauce** The magic ingredient that sets Apache Spark apart from many other data processing frameworks is its effective utilization of in-memory processing. In-memory processing involves storing and processing data directly in a computer's RAM (Random Access Memory), as opposed to reading and writing data to slower storage devices like hard drives. Source: https://medium.com/@harshavardhan7696/unleashing-speed-and-efficiency-in-memory-processing-in-apache-spark-c9e7ccc6460e. The RDDs contain variables that allow the nodes to process data in parallel. RDDs are processed using functions and variables that can be used to create other RDDs to further process the data set.

In-Memory Computation

Spark uses in-memory computation as a way to speed up the total processing time. In the in-memory

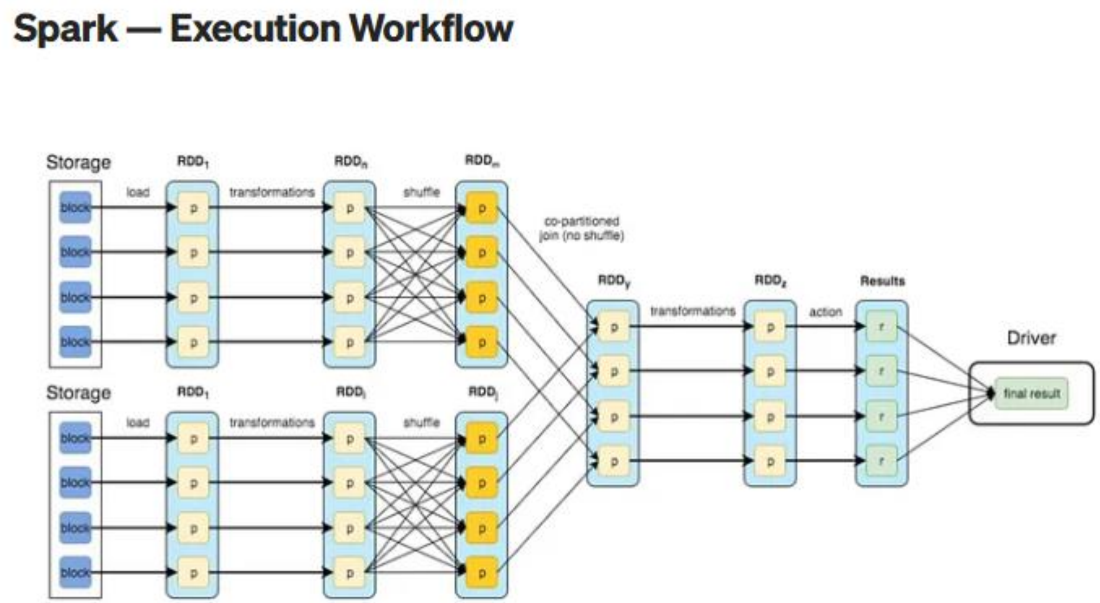
[redacted]

U.S. Patent No. 7,712,080 (Claim 9)	
Claim(s)	**Example Southwest Count VII Systems and Services**
	RDD Creation Here's an example of RDDs created during a method call: Which first loads HDFS blocks in memory and then applies map() function to filter out keys creating two RDDs:  Source: https://blog.k2datascience.com/batch-processing-apache-spark-a67016008167.
[9.b] at least one distributed sequential computing program, configured to operate in the at least one processor, configured to access the at least one distributed shared variable,	On information and belief, the Southwest Count VII Systems and Services include at least one distributed sequential computing program configured to operate in the at least one processor, access the at least one distributed shared variable, and transform into at least one distributed parallel computing program by spawning at least one child distributed sequential computing system program when at least one intermediate condition occurs within the at least one distributed sequential program.

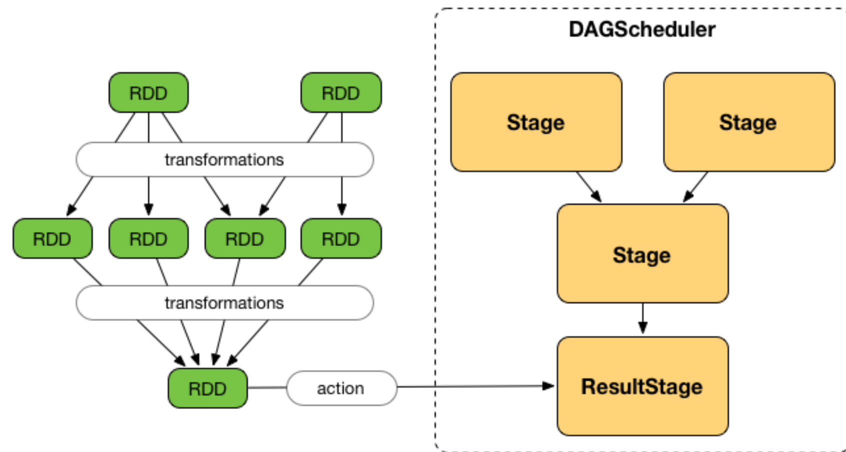
[redacted]

U.S. Patent No. 7,712,080 (Claim 9)	
Claim(s)	**Example Southwest Count VII Systems and Services**
and configured to transform into at least one distributed parallel computing program by spawning at least one child distributed sequential computing system program when at least one intermediate condition occurs within the at least one distributed sequential program,	For example, Spark contains a driver program that is configured to operate in the at least one processor and is configured to access the RDDs. Spark jobs are authored in programming languages such as Java, Scala, and Python, and Spark is configured to access the at least one distributed shared variable (RDDs) through the access of RDDs by a Spark job. **Spark Overview** Apache Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming. Source: https://archive.apache.org/dist/spark/docs/2.0.1/. **Overview** At a high level, every Spark application consists of a *driver program* that runs the user's `main` function and executes various *parallel operations* on a cluster. The main abstraction Spark provides is a *resilient distributed dataset* (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it. Users may also ask Spark to *persist* an RDD in memory, allowing it to be reused efficiently across parallel operations. Finally, RDDs automatically recover from node failures. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html. The Spark Execution Model transforms the distributed sequential computing program into a series of stages and sets of parallel tasks that run in parallel on the Spark cluster. This includes Spark tasks operating on a set of RDDs in parallel for a given stage.

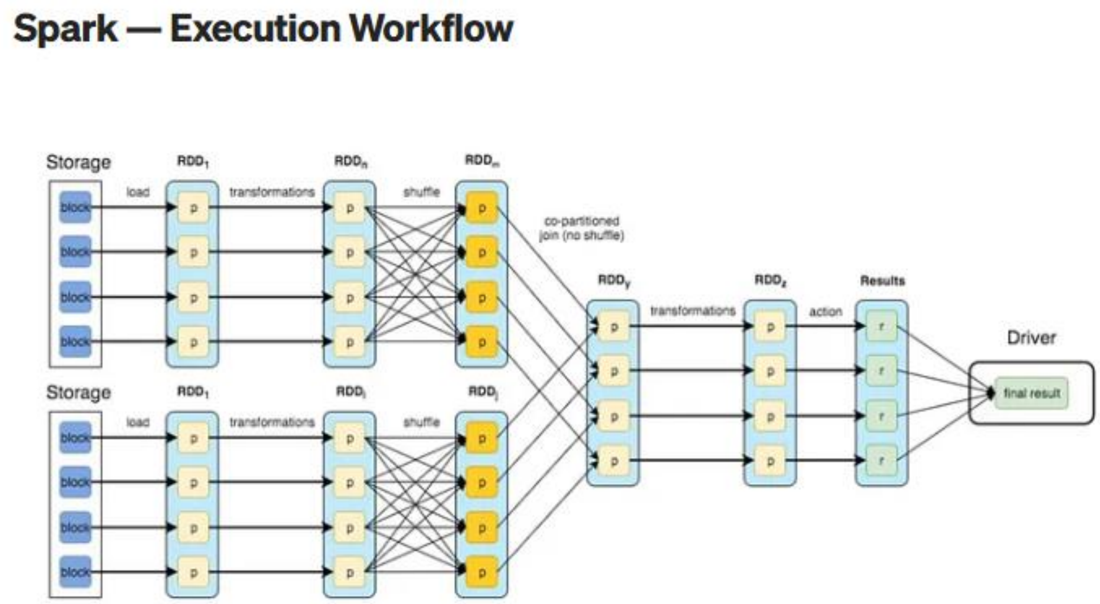
[redacted]

U.S. Patent No. 7,712,080 (Claim 9)	
Claim(s)	**Example Southwest Count VII Systems and Services**
	 Source: https://blog.k2datascience.com/batch-processing-apache-spark-a67016008167. Spark implements shuffle boundaries where stages and/or tasks wait for the prior stage to complete before fetching map outputs.

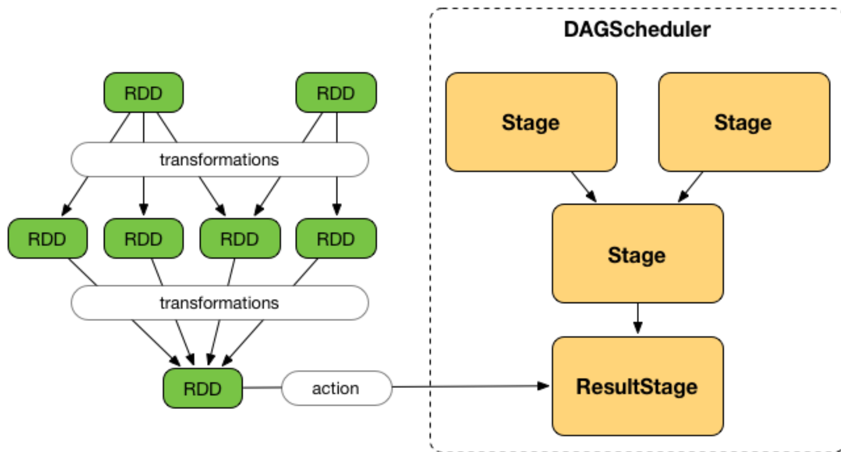
[redacted]

U.S. Patent No. 7,712,080 (Claim 9)	
Claim(s)	**Example Southwest Count VII Systems and Services**
	DAGScheduler splits up a job into a collection of stages. Each stage contains a sequence of narrow transformations that can be completed without shuffling the entire data set, separated at **shuffle boundaries**, i.e. where shuffle occurs. Stages are thus a result of breaking the RDD graph at shuffle boundaries.  Figure 4. Graph of Stages Shuffle boundaries introduce a barrier where stages/tasks must wait for the previous stage to finish before they fetch map outputs. Source: https://mallikarjuna_g.gitbooks.io/spark/content/spark-dagscheduler-stages.html.
[9.b.iv] wherein the at least one distributed parallel computing program concurrently uses the at least one distributed sequential computing program and the at least one spawned child distributed sequential computing program to	On information and belief, the Southwest Count VII Systems and Services include the at least one distributed parallel computing program concurrently uses the at least one distributed sequential computing program and the at least one spawned child distributed sequential computing program to perform parallel processing and/or operations. For example, Spark implements a set of tasks running in parallel for a given stage. The distributed parallel computing program includes the Spark job and driver program, as described in this claim chart. The spawned child distributed sequential computing program is the sequential code in a task for a given stage, as described in this claim chart. *See* claim limitations [9.b] and [9.b.i-iii].

[redacted]

U.S. Patent No. 7,712,080 (Claim 9)	
Claim(s)	**Example Southwest Count VII Systems and Services**
perform parallel processing and/or operations,	 Source: https://blog.k2datascience.com/batch-processing-apache-spark-a67016008167.
[9.b.v] wherein the at least one intermediate condition comprising one intermediate result that will be required by the at least one spawned child distributed sequential computing program to continue computation.	On information and belief, the Southwest Count VII Systems and Services include the at least one intermediate condition comprising one intermediate result that will be required by the at least one spawned child distributed sequential computing program to continue computation. For example, Spark implements shuffle boundary where stages and/or tasks wait for the prior stage to complete before fetching map outputs.

[redacted]

U.S. Patent No. 7,712,080 (Claim 9)	
Claim(s)	**Example Southwest Count VII Systems and Services**
	DAGScheduler splits up a job into a collection of stages. Each stage contains a sequence of narrow transformations that can be completed without shuffling the entire data set, separated at **shuffle boundaries**, i.e. where shuffle occurs. Stages are thus a result of breaking the RDD graph at shuffle boundaries.  Figure 4. Graph of Stages Shuffle boundaries introduce a barrier where stages/tasks must wait for the previous stage to finish before they fetch map outputs. Source: https://mallikarjuna_g.gitbooks.io/spark/content/spark-dagscheduler-stages.html.